Overview

Use this document as a general set of performance guidelines for creating assets for use in CRYENGINE. By following the tips included below, artists will better understand how to create assets, with the performance in mind.

Importance of Asset Optimization

Typically, objects that are always on screen go through a lot of iterations and need to have a higher visual quality than the other assets. This also means that these objects need to be as performance friendly as possible. You will need to review all the characters and weapons with this in mind.

In addition, AI and particle effects play an important role. These things tend to turn heavy areas into areas with severe performance problems. Thus, it is important to identify these areas first and make all their related art assets as performance friendly as possible.

Scene Complexity

If you want to hit a specific frame rate, the total amount of vertices and texture space on screen at the same time needs to be consistent for every level. Given the budget that a developer chooses, there needs to be a difference in scene complexity between heavily occluded indoor environments that could use portals, and more un-occluded outdoor environments. The smaller the visibility, the more detail you can put in the level.

Some techniques to make sure you reach the performance goals from the level design side are:

- Close off rooms with doors.
- Use transition zones.
- Use fog effects for un-occluded environments.
- Use Visblocking/occluders with large objects.
- Specifically place heavy objects in areas where the visibility spectrum is small.
- Delete terrain where it is not visible to the player.
- Smooth terrain to decrease vertex density.
- Adjust the view distance ratio for objects.
- Have a maximum of 2000 drawcalls, and for most projects, even less.
- Use as few textures on the terrain as possible.
- Avoid overlapping several materials.
- Don't overlay several, very large transparent or additive layers.
- Keep in touch with artists regarding LOD distance ratios of assets.

Taking these into account and adjusting some of these numbers, we can arrive at a breakdown even it is quite broad it will give a general idea.

Keep in mind that the DP's (drawpoints/drawcalls) were deliberately kept below 2000 to allow a bit of breathing room for gameplay action firing effects and additional polishing:

Object	Triangles	Drawcalls
Terrain	50000	80
Particles	2000	100
Roads	30000	120
Decals	20000	80
Veg	40000	200
Brushes	600000	1000
Entities	300000	320
FP Character	75000	40

TOTAL	**1042000**	**1940**

CPU Performance

There are ways to improve CPU performance by using fewer draw calls. Do this by using fewer materials, attachments, sprites (instead of LODs), lights (more objects combined on one texture), decals, and shadow casting lights.

Some additional guidelines for improving the CPU performance are:

- Use fewer complicated materials (cheaper setup cost).
- Use fewer triangles in the cbuffer (saves draw calls, conflicts with less draw calls).
- Limit the area covered in the cbuffer.
- Use fewer bones for animated characters.
- Use less alpha blended objects in atmospheric effects (CPU fog approximation).
- Use fewer triangles in the physics proxy (hit detection).
- Use breakability sparingly.

GPU Performance

The following recommendations help improve GPU performance, regardless of the screen resolution:

- Use fewer rendered vertices (a vertex is a unique point with position and texture coordinates).
- Use a less complex vertex format (mostly a PS3 issue as more complex formats are required for vegetation animation and bone skinning).
- Use fewer shadow casting lights.
- Use decals sparingly.

The following recommendations help improve the GPU performance, dependent on the screen resolution.

- Render objects that cover less pixels on the screen.
- Use cheaper pixel shaders (less lights, less shader features like reflection, simpler shader); the most simple *Illum* shader should be used.
- Use fewer textures for the scene and for each material.
- Use cheap texture formats (should be compressed like DXT1, DXT5, 3Dc/DXT5).
- Alphablend is slower than alphatest, which in turn is much slower than normal opaque rendering.
- Have fewer post-processing effects enabled at the same time (example: if radial blurring is enabled, try to not use any more post effects). Another strategy would be to decrease the amount of time a post effect is visible on screen.
- Use no or little anti-aliasing.
- Use fewer atmospheric effects.

Texture Streaming

To improve texture streaming, follow these guidelines:

- All streamable textures should be square in size (equal in width and height: 1024 x 1024, 256 x 256, etc) for better management of the texture pools.
- Use as few different texture formats as possible for better management of the texture pools.
- Use as few unique textures per area as possible, to reduce the I/O bandwidth.
- Stick rigidly to the scheduled *texels per square meter* (distant streaming is based on it), and use **r_texelsPerMeter 1** to check it.
- Try to avoid using textures smaller than 128 x 128 (it causes memory fragmentation). Combine them into a single texture for the same area.

General Performance Guidelines

This section includes general tips for improving the performance.

- Less materials can be used by using textures wisely.
- Attachments can be tweaked to disappear when small on the screen.
- Eyes can be closed at distance to avoid the need for eyeball rendering.
- Destructible objects can be heavy on performance.
- Unwrapping of objects should be made with as few vertices (big pieces in UV).
- Vis areas help to reduce the object count.
- Portals are more efficient than cbuffer for static scenes.
- Blocky geometry helps to get more for the same vertex count.
- Reuse geometry/textures to save memory.
- Material layers that require multi-pass rendering, which is slower than normal rendering, should be used sparingly.
- Either apply Reset Xform or reset transformations manually (rotation/scaling) prior to export - these values are picked up by the engine and can generate issues such as the physics not matching with render geometry.
- Follow and fix all warnings/errors that appear in the level error window or console of the editor.

Level design setup / splitting large meshes

This section deals with the performance-oriented asset and level design setup of very large meshes, such as buildings.

Large objects, like buildings, should be split up in a way that enables them to be rendered efficiently in standard gameplay situations. This means that the lower part of a skyscraper, an area the player interacts with and which usually lies in your viewing angle when looking straight ahead, should be in a different CGF than the upper part of the building, which the player only sees when looking straight up. Big, complex parts of a building that are out of sight in the normal way you play a level, such as a building that can only be seen from one side when approached through a street, should always be detached and not be part of the main mesh.

This also helps effective LOD'ing, since the engine decides the LOD level of an object based on it's distance to the camera and size of its bounding box. Very large meshes are switching LOD's much later smaller ones. The effectiveness of occlusion culling is also greatly enhanced by this.
For this reason, it's sometimes beneficial to split off certain parts of a building and adjust the LOD ratio. Initially, the game might have to process more drawcalls if you do this, but this is compensated by the more effective culling and LOD'ing that becomes possible with such an approach.

Some practical tips on working in such a way are:

- Aligning the pivot of the detached part to the main part of the mesh, to ensure that they can be placed in the level with precision, and avoiding gaps between the two parts. Alternatively, the pivots can be different, but aligned into the grid inside 3dsmax, so they can be placed along the grid inside the editor for the same effect. Note that there are certain float rounding errors and imprecision issues in the engine which are hard to avoid, especially if the area in the level is offset far from the level's origin (0,0,0). If there are visible gaps due to this, work around this by splitting the building along a structure that hides the gap, such as a trim going around the facade.
- Even though it creates some extra unnecessary vertices, the slice modifier in 3dsmax is useful for such setups, as it is non-destructive and allows later changes to where you split the meshes (especially relevant with LOD's - slice modifiers can be instanced).
- Note that each part of the split mesh needs to have its correct collision meshes assigned. By assigning all collision meshes to one brush only, the player collision will still work if both parts are placed next to each other, but discrete collision geometry is still needed for each brush due to decal placement and refined collisions with grenades and other physical objects.
- Try to use different sub-materials for the separate parts, and using as few different ones as possible per piece, so that the drawcalls don't necessarily double when splitting up an object. If possible, plan far ahead on how large objects are placed in the level and seen by the player, in order to build them as effective as possible.

Drawcalls

Every object with a different material has to be drawn in a separate drawcall. Each drawcall adds some overhead / slowdown to the engine. At any given point of view in the level, there should never be more than 2000 drawcalls.

Several examples for reducing the number of draw calls:

- Where it makes sense, merge some textures together (for example if you have two different 512x256 textures used in the same object, it would be better to merge them into a single 512x512 texture).
- Square textures are more stream friendly.
- Use as little different materials per object as possible. Each material in an object creates at least 2 draw calls, with a shadows being enabled 3 draw calls are needed.

What else adds drawcalls?

The following list details how the number of drawcalls used is calculated:

- Each material (at least 2 drawcalls).
  - ZPass, Opaque pass (general pass).
  - Various shader features per-material can cost additional drawcalls.
- Sub-materials also cost additional drawcalls.
- Lights and shadows.
- Post processes.
- HUD, text rendering, and the like.

Material Properties

The following are the material properties that add to the drawcall count:

- Glow.
- Detail mapping.
- Detail decals.
- Scattering.
- Decals.
  - Using normal maps adds an extra drawcall on ZPass.
- Terrain
  - Base pass + detail layers is at least 2 drawcalls.
  - The more the number of layers, the more the drawcalls (and overdraw).

Number of Vertices

Having less vertices will improve performance and reduce memory consumption. On some hardware like PS3 and older graphics cards, the cost of vertex processing is quite expensive.

- UV borders increase the vertex count in the engine - use as little UV borders as possible.
- Vertex normals need to be split where polys with different smoothing groups(hard edges) meet - keep the number of smoothing groups and hard edges low.

Physics Proxy

A good physics proxy is very important for reducing CPU calculation time for physics. Without a physics proxy, performance would be too slow.

- Use a simple physics proxy instead of the render geometry for physics calculation, in order to reduce CPU calculation time for physics.
- the render geometry should only be used for physics calculation for 3d breakable objects like tree trunks.
- If possible use primitives as physics proxies.
- Make use of the possibility of rough physics proxies for player collisions and render geometry checks for bullet collision checks.
- Use 1 smoothing group (only soft edges) per physics proxy.
- Set the proper physics type in the material editor of the 3D package.

Level of Detail Objects (LOD's)

Always make LODs for each asset. Each LOD must reduce polygon count at least by 50%. For each LOD, do not only reduce the polygon count, but try reducing the number of materials as well. Good LOD's are key to getting good performance (less vertices to process, less materials / drawcalls etc. will improve performance a lot).

Good LOD's are key for getting good performance (less vertices to process, less materials / draw calls etc. will improve performance a lot). If you create good LOD's, you could get away with better looking assets and better performance.

- Always create LOD's for art assets, even in the testing phase. Make use of instancing and poly reduction modifiers in 3ds Max.
- Each LOD step must reduce the polygon count at least by 50%.
- Reduce materials in LOD's - this saves on draw calls, just add additional material ID's in the material and use them only in the LOD's.

Material Setup

- Use the fewest number of materials as possible.
- Combine textures to reduce materials and texture fetches.
- Keep all textures in CryTIF format.
- Use 187/187/187 (128 + gamma correction) as a default diffuse color (adjustments in either direction can then be made more easily).
- Use simple grey (128/128/128) or quickly colored textures in final size for early tests - to get an idea about memory consumption.

Vegetation

In outdoor games, vegetation is usually the most expensive part of the level due to the fact that natural levels are populated with many trees.

- Reduce the materials for vegetation objects as much as possible to reduce number of draw calls.
- Use only the optimized vegetation shader on vegetation objects.
- Spend a lot of time to reduce the number of polygons - these objects are drawn multiple times on screen.
- Combine textures in one and reduce the number of materials or at least texture fetches.
- Add polygons on the leaves to match the texture - this reduces overdraw (use the triangles you saved on the trunk).

Particles

Make particle effects as effective as possible, as they create a huge amount of overdraw and thus have a big effect on performance. Also, place standing particle effects as early as possible in order to get a feeling for the performance impact.